Des programmes sans bugs grâce aux mathématiques formelles

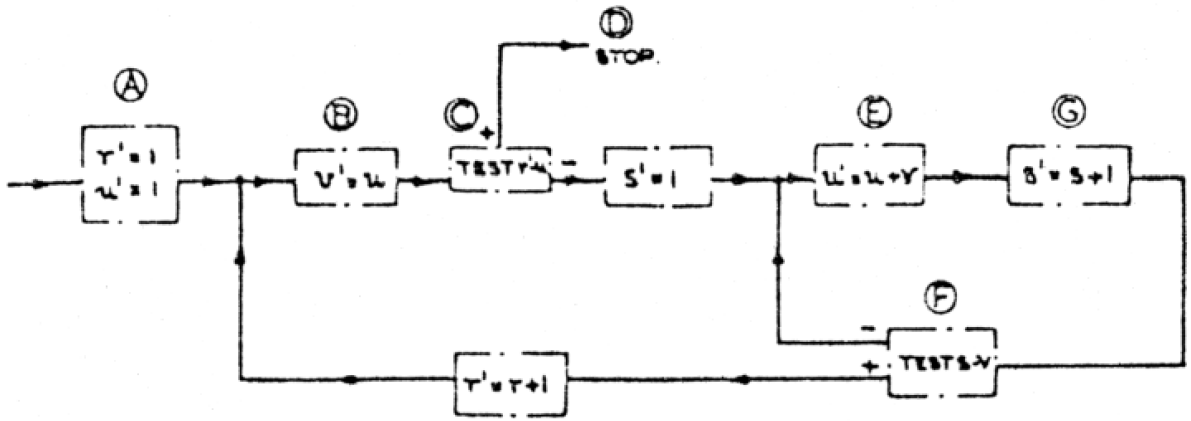
Jean-Marie Madiot

INRIA

2 décembre 2023

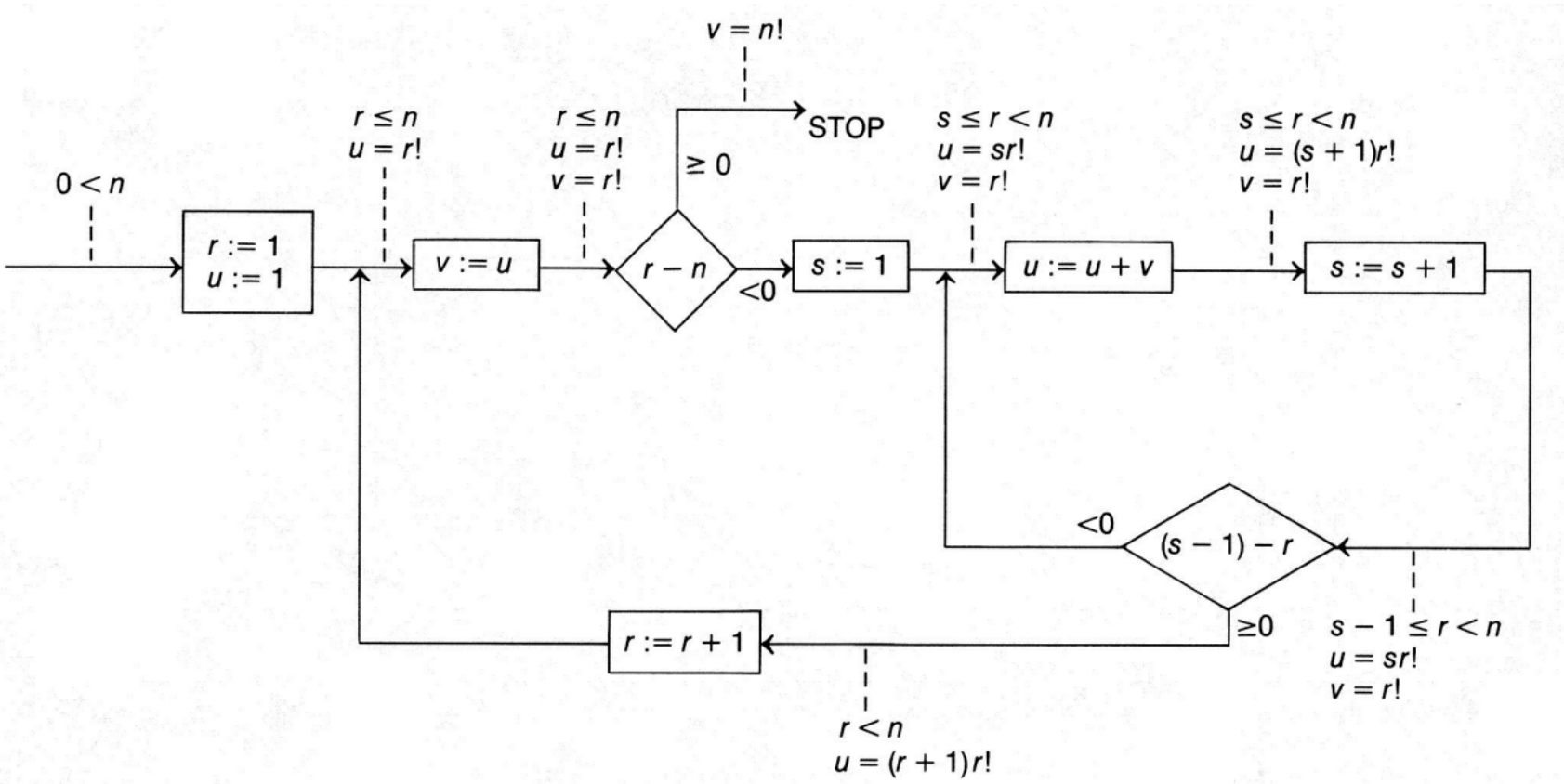
Alan Turing, 1949 : *Checking a large routine*

"a small routine to obtain $n!$ without the use of a multiplier"



STORAGE LOCATION	(INITIAL) Ⓐ k=6	Ⓑ k=5	Ⓒ k=4	(STOP) Ⓓ k=0	Ⓔ k=3	Ⓕ k=1	Ⓖ k=2
27					s	$s+1$	s
28		r	r		r	r	r
29	n	n	n	n	r	n	n
30		Δ	Δ		$s\Delta$	$(s+1)\Delta$	$(s+1)\Delta$
31			Δ	Δ	Δ	Δ	Δ
	TO Ⓑ WITH $r'=1$ $s'=1$	TO Ⓒ	TO Ⓓ IF $r=n$ TO Ⓔ IF $r<n$		TO Ⓖ	TO Ⓑ WITH $r'=r+1$ IF $s \geq r$ TO Ⓔ WITH $r'=s+1$ IF $s<r$	TO Ⓕ

Plus lisible: Morris, Jones, 1984 : *An Early Program Proof by Alan Turing*



L'argument de Turing : pas besoin d'avoir tout le programme en tête. Il suffit de vérifier pour chaque boîte la cohérence entre :

- la **précondition** (annotation entrante)
- l'**action** de l'instruction
- la **postcondition** (annotation sortante)

Présentation plus moderne

Code plus structuré (pas de flèche/GOTO), moins d'annotations :

- fonctions, avec leurs pré/postconditions
- boucles, avec les *invariants de boucle*

```
def fact(n):
  # requires n >= 0, returns n!
  i = 1
  x = 1
  while i < n:
    # invariant: i <= n,  x = i!
    j = 1
    y = x
    while j <= i:
      # invariant: j - 1 <= i < n,  y = j * i!,  x = i!
      y = y + x
      j = j + 1
    i = i + 1
    x = y
  return x
```

Bon bah c'est plié ?

Oui... on *peut* faire des preuves à la main, mais :

- facile mais pénible, nombreux cas, grosses formules

Bon bah c'est plié ?

Oui... on *peut* faire des preuves à la main, mais :

- facile mais pénible, nombreux cas, grosses formules
- à refaire quand le code change (fréquent lors des preuves)

Bon bah c'est plié ?

Oui... on *peut* faire des preuves à la main, mais :

- facile mais pénible, nombreux cas, grosses formules
- à refaire quand le code change (fréquent lors des preuves)
- ignorance/oubli de subtilités selon le langage :
  - 2^{63} : positif ? négatif ? nul ?

Bon bah c'est plié ?

Oui... on *peut* faire des preuves à la main, mais :

- facile mais pénible, nombreux cas, grosses formules
- à refaire quand le code change (fréquent lors des preuves)
- ignorance/oubli de subtilités selon le langage :
  - 2^{63} : positif ? négatif ? nul ?
  - `x + (x = 1);` interdit ou non-déterministe

Bon bah c'est plié ?

Oui... on *peut* faire des preuves à la main, mais :

- facile mais pénible, nombreux cas, grosses formules
- à refaire quand le code change (fréquent lors des preuves)
- ignorance/oubli de subtilités selon le langage :
  - 2^{63} : positif ? négatif ? nul ?
  - `x + (x = 1);` interdit ou non-déterministe
- erreurs parfois coûteuses, voire dangereuses

Bon bah c'est plié ?

Oui... on *peut* faire des preuves à la main, mais :

- facile mais pénible, nombreux cas, grosses formules
- à refaire quand le code change (fréquent lors des preuves)
- ignorance/oubli de subtilités selon le langage :
  - 2^{63} : positif ? négatif ? nul ?
  - `x + (x = 1);` interdit ou non-déterministe
- erreurs parfois coûteuses, voire dangereuses

Problème mathématique moins risqué et plus intéressant :
comment formaliser ça pour que l'ordinateur vérifie tout ?

(reste de l'exposé : faisable en Coq, Lean, etc)

Le langage “While” ou “IMP”

Un langage jouet impératif avec des variables (x), des entiers ($n \in \mathbb{Z}$), des expressions arithmétiques et booléens, et surtout des boucles while :

$$
\begin{array}{lll}
e & ::= & x \mid n \mid e_1 + e_2 \mid e_1 - e_2 \mid e_1 * e_2 \mid e_1 \leqslant e_2 \mid e_1 \wedge e_2 \mid \neg e \\
s & ::= & \texttt{skip} \mid x := e \mid s_1; s_2 \mid \texttt{if } e \texttt{ then } s_1 \texttt{ else } s_2 \mid \texttt{while } e \texttt{ do } s
\end{array}
$$

Le langage “While” ou “IMP”

Un langage jouet impératif avec des variables (x), des entiers ($n \in \mathbb{Z}$), des expressions arithmétiques et booléens, et surtout des boucles while :

$$
\begin{array}{lll}
e & ::= & x \mid n \mid e_1 + e_2 \mid e_1 - e_2 \mid e_1 * e_2 \mid e_1 \leqslant e_2 \mid e_1 \wedge e_2 \mid \neg e \\
s & ::= & \texttt{skip} \mid x := e \mid s_1; s_2 \mid \texttt{if } e \texttt{ then } s_1 \texttt{ else } s_2 \mid \texttt{while } e \texttt{ do } s
\end{array}
$$

Exemples :

`skip` ne fait rien

Le langage “While” ou “IMP”

Un langage jouet impératif avec des variables (x), des entiers ($n \in \mathbb{Z}$), des expressions arithmétiques et booléens, et surtout des boucles while :

$$
\begin{array}{lll}
e & ::= & x \mid n \mid e_1 + e_2 \mid e_1 - e_2 \mid e_1 * e_2 \mid e_1 \leqslant e_2 \mid e_1 \wedge e_2 \mid \neg e \\
s & ::= & \texttt{skip} \mid x := e \mid s_1; s_2 \mid \texttt{if } e \texttt{ then } s_1 \texttt{ else } s_2 \mid \texttt{while } e \texttt{ do } s
\end{array}
$$

Exemples :

`skip`	ne fait rien
$x := 1; x := 2 * x$	calcule $x = 2$

Le langage “While” ou “IMP”

Un langage jouet impératif avec des variables (x), des entiers ($n \in \mathbb{Z}$), des expressions arithmétiques et booléens, et surtout des boucles while :

$$
\begin{array}{lll}
e & ::= & x \mid n \mid e_1 + e_2 \mid e_1 - e_2 \mid e_1 * e_2 \mid e_1 \leqslant e_2 \mid e_1 \wedge e_2 \mid \neg e \\
s & ::= & \texttt{skip} \mid x := e \mid s_1; s_2 \mid \texttt{if } e \texttt{ then } s_1 \texttt{ else } s_2 \mid \texttt{while } e \texttt{ do } s
\end{array}
$$

Exemples :

`skip`	ne fait rien
$x := 1; x := 2 * x$	calcule $x = 2$
`if` $x > 0$ `then` $r := x$ `else` $r := 0 - x$	calcule $r = \lvert x \rvert$

Le langage “While” ou “IMP”

Un langage jouet impératif avec des variables (x), des entiers ($n \in \mathbb{Z}$), des expressions arithmétiques et booléens, et surtout des boucles while :

$$
\begin{array}{lll}
e & ::= & x \mid n \mid e_1 + e_2 \mid e_1 - e_2 \mid e_1 * e_2 \mid e_1 \leqslant e_2 \mid e_1 \wedge e_2 \mid \neg e \\
s & ::= & \texttt{skip} \mid x := e \mid s_1; s_2 \mid \texttt{if } e \texttt{ then } s_1 \texttt{ else } s_2 \mid \texttt{while } e \texttt{ do } s
\end{array}
$$

Exemples :

$\texttt{skip}$	ne fait rien
$x := 1; x := 2 * x$	calcule $x = 2$
$\texttt{if } x > 0 \texttt{ then } r := x \texttt{ else } r := 0 - x$	calcule $r = \lvert x \rvert$
$\texttt{while } n \neq 0 \texttt{ do } n := n - 1$	calcule 0 parfois

Le langage “While” ou “IMP”

Un langage jouet impératif avec des variables (x), des entiers ($n \in \mathbb{Z}$), des expressions arithmétiques et booléens, et surtout des boucles while :

$$
\begin{array}{lll}
e & ::= & x \mid n \mid e_1 + e_2 \mid e_1 - e_2 \mid e_1 * e_2 \mid e_1 \leqslant e_2 \mid e_1 \wedge e_2 \mid \neg e \\
s & ::= & \texttt{skip} \mid x := e \mid s_1; s_2 \mid \texttt{if } e \texttt{ then } s_1 \texttt{ else } s_2 \mid \texttt{while } e \texttt{ do } s
\end{array}
$$

Exemples :

$\texttt{skip}$	ne fait rien
$x := 1; x := 2 * x$	calcule $x = 2$
$\texttt{if } x > 0 \texttt{ then } r := x \texttt{ else } r := 0 - x$	calcule $r = \lvert x \rvert$
$\texttt{while } n \neq 0 \texttt{ do } n := n - 1$	calcule 0 parfois
$r := 1; \texttt{while } n > 0 \texttt{ do } (r := r * x; n := n - 1)$	calcule $r = x^n$

Le langage “While” ou “IMP”

Un langage jouet impératif avec des variables (x), des entiers ($n \in \mathbb{Z}$), des expressions arithmétiques et booléens, et surtout des boucles while :

$$
\begin{array}{lll}
e & ::= & x \mid n \mid e_1 + e_2 \mid e_1 - e_2 \mid e_1 * e_2 \mid e_1 \leqslant e_2 \mid e_1 \wedge e_2 \mid \neg e \\
s & ::= & \texttt{skip} \mid x := e \mid s_1; s_2 \mid \texttt{if } e \texttt{ then } s_1 \texttt{ else } s_2 \mid \texttt{while } e \texttt{ do } s
\end{array}
$$

Exemples :

$\texttt{skip}$	ne fait rien
$x := 1; x := 2 * x$	calcule $x = 2$
$\texttt{if } x > 0 \texttt{ then } r := x \texttt{ else } r := 0 - x$	calcule $r = \lvert x \rvert$
$\texttt{while } n \neq 0 \texttt{ do } n := n - 1$	calcule 0 parfois
$r := 1; \texttt{while } n > 0 \texttt{ do } (r := r * x; n := n - 1)$	calcule $r = x^n$
$x := 0; s := 1; \texttt{while } s \leqslant n \texttt{ do } x := x + 1; s := s + 2 * x + 1$	calcule $x = \lfloor \sqrt{n} \rfloor$

Logique de Hoare (Floyd 1967, Hoare 1969)

Définition (triplet de Hoare) on note $\{P\}\ s\ \{Q\}$ pour :

"Si P est vrai et qu'on lance le programme s, à la fin on aura Q"

Logique de Hoare (Floyd 1967, Hoare 1969)

Définition (triplet de Hoare) on note $\{P\}\ s\ \{Q\}$ pour :

"Si P est vrai et qu'on lance le programme s, à la fin on aura Q"

Règles de la logique :

$$\frac{}{\{P\}\ \texttt{skip}\ \{P\}}$$

Logique de Hoare (Floyd 1967, Hoare 1969)

Définition (triplet de Hoare) on note $\{P\}\ s\ \{Q\}$ pour :

"Si P est vrai et qu'on lance le programme s, à la fin on aura Q"

Règles de la logique :

$$\frac{}{\{P\}\ \texttt{skip}\ \{P\}} \qquad \frac{}{\{P[e/x]\}\ x := e\ \{P\}}$$

Logique de Hoare (Floyd 1967, Hoare 1969)

Définition (triplet de Hoare) on note $\{P\}\ s\ \{Q\}$ pour :

"Si P est vrai et qu'on lance le programme s, à la fin on aura Q"

Règles de la logique :

$$\frac{}{\{P\}\ \texttt{skip}\ \{P\}} \qquad \frac{}{\{x+1=1\}\ x := x+1\ \{x=1\}}$$

Logique de Hoare (Floyd 1967, Hoare 1969)

Définition (triplet de Hoare) on note $\{P\}\ s\ \{Q\}$ pour :

"Si P est vrai et qu'on lance le programme s, à la fin on aura Q"

Règles de la logique :

$$\frac{}{\{P\}\ \texttt{skip}\ \{P\}} \qquad \frac{}{\{P[e/x]\}\ x := e\ \{P\}}$$

Logique de Hoare (Floyd 1967, Hoare 1969)

Définition (triplet de Hoare) on note $\{P\}\ s\ \{Q\}$ pour :

"Si P est vrai et qu'on lance le programme s, à la fin on aura Q"

Règles de la logique :

$$\frac{}{\{P\}\ \texttt{skip}\ \{P\}} \qquad \frac{}{\{P[e/x]\}\ x := e\ \{P\}}$$

$$\frac{\{P \wedge B\}\ s_1\ \{Q\} \quad \{P \wedge \neg B\}\ s_2\ \{Q\}}{\{P\}\ \texttt{if}\ B\ \texttt{then}\ s_1\ \texttt{else}\ s_2\ \{Q\}}$$

Logique de Hoare (Floyd 1967, Hoare 1969)

Définition (triplet de Hoare) on note $\{P\}\ s\ \{Q\}$ pour :

"Si P est vrai et qu'on lance le programme s, à la fin on aura Q"

Règles de la logique :

$$\frac{}{\{P\}\ \texttt{skip}\ \{P\}} \qquad \frac{}{\{P[e/x]\}\ x := e\ \{P\}}$$

$$\frac{\{P \wedge B\}\ s_1\ \{Q\} \quad \{P \wedge \neg B\}\ s_2\ \{Q\}}{\{P\}\ \texttt{if}\ B\ \texttt{then}\ s_1\ \texttt{else}\ s_2\ \{Q\}} \qquad \frac{\{P\}\ s_1\ \{Q\} \qquad \{Q\}\ s_2\ \{R\}}{\{P\}\ s_1; s_2\ \{R\}}$$

Logique de Hoare (Floyd 1967, Hoare 1969)

Définition (triplet de Hoare) on note $\{P\}\ s\ \{Q\}$ pour :

"Si P est vrai et qu'on lance le programme s, à la fin on aura Q"

Règles de la logique :

$$\frac{}{\{P\}\ \texttt{skip}\ \{P\}} \qquad \frac{}{\{P[e/x]\}\ x := e\ \{P\}}$$

$$\frac{\{P \wedge B\}\ s_1\ \{Q\} \quad \{P \wedge \neg B\}\ s_2\ \{Q\}}{\{P\}\ \texttt{if}\ B\ \texttt{then}\ s_1\ \texttt{else}\ s_2\ \{Q\}} \qquad \frac{\{P\}\ s_1\ \{Q\} \qquad \{Q\}\ s_2\ \{R\}}{\{P\}\ s_1; s_2\ \{R\}}$$

$$\frac{\{I \wedge B\}\ s\ \{I\}}{\{I\}\ \texttt{while}\ B\ \texttt{do}\ s\ \{I \wedge \neg B\}}$$

Logique de Hoare (Floyd 1967, Hoare 1969)

Définition (triplet de Hoare) on note $\{P\}\ s\ \{Q\}$ pour :

"Si P est vrai et qu'on lance le programme s, à la fin on aura Q"

Règles de la logique :

$$\frac{}{\{P\}\ \texttt{skip}\ \{P\}} \qquad \frac{}{\{P[e/x]\}\ x := e\ \{P\}}$$

$$\frac{\{P \wedge B\}\ s_1\ \{Q\} \quad \{P \wedge \neg B\}\ s_2\ \{Q\}}{\{P\}\ \texttt{if}\ B\ \texttt{then}\ s_1\ \texttt{else}\ s_2\ \{Q\}} \qquad \frac{\{P\}\ s_1\ \{Q\} \qquad \{Q\}\ s_2\ \{R\}}{\{P\}\ s_1; s_2\ \{R\}}$$

$$\frac{\{I \wedge B\}\ s\ \{I\}}{\{I\}\ \texttt{while}\ B\ \texttt{do}\ s\ \{I \wedge \neg B\}} \qquad \frac{P \Rightarrow P' \quad \{P'\}\ s\ \{Q'\} \quad Q' \Rightarrow Q}{\{P\}\ s\ \{Q\}}$$

Logique de Hoare (Floyd 1967, Hoare 1969)

Définition (triplet de Hoare) on note $\{P\}\ s\ \{Q\}$ pour :

"Si P est vrai et qu'on lance le programme s, à la fin on aura Q"

Règles de la logique :

$$\frac{}{\{P\}\ \texttt{skip}\ \{P\}} \qquad \frac{}{\{P[e/x]\}\ x := e\ \{P\}}$$

$$\frac{\{P \wedge B\}\ s_1\ \{Q\} \quad \{P \wedge \neg B\}\ s_2\ \{Q\}}{\{P\}\ \texttt{if}\ B\ \texttt{then}\ s_1\ \texttt{else}\ s_2\ \{Q\}} \qquad \frac{\{P\}\ s_1\ \{Q\} \qquad \{Q\}\ s_2\ \{R\}}{\{P\}\ s_1; s_2\ \{R\}}$$

$$\frac{\{I \wedge B\}\ s\ \{I\}}{\{I\}\ \texttt{while}\ B\ \texttt{do}\ s\ \{I \wedge \neg B\}} \qquad \frac{P \Rightarrow P' \quad \{P'\}\ s\ \{Q'\} \quad Q' \Rightarrow Q}{\{P\}\ s\ \{Q\}}$$

Exemple : souvent pratique de partir de la fin :

$$\frac{\{\qquad\qquad\}\ x := x + 1\ \{\qquad\qquad\} \qquad \{\qquad\qquad\}\ x := 2 * x\ \{\qquad\qquad\}}{\{\qquad\qquad\}\ x := x + 1; x = 2 * x\ \{x > 6\}}$$

Logique de Hoare (Floyd 1967, Hoare 1969)

Définition (triplet de Hoare) on note $\{P\}\ s\ \{Q\}$ pour :

"Si P est vrai et qu'on lance le programme s, à la fin on aura Q"

Règles de la logique :

$$\frac{}{\{P\}\ \texttt{skip}\ \{P\}} \qquad \frac{}{\{P[e/x]\}\ x := e\ \{P\}}$$

$$\frac{\{P \wedge B\}\ s_1\ \{Q\} \quad \{P \wedge \neg B\}\ s_2\ \{Q\}}{\{P\}\ \texttt{if}\ B\ \texttt{then}\ s_1\ \texttt{else}\ s_2\ \{Q\}} \qquad \frac{\{P\}\ s_1\ \{Q\} \qquad \{Q\}\ s_2\ \{R\}}{\{P\}\ s_1; s_2\ \{R\}}$$

$$\frac{\{I \wedge B\}\ s\ \{I\}}{\{I\}\ \texttt{while}\ B\ \texttt{do}\ s\ \{I \wedge \neg B\}} \qquad \frac{P \Rightarrow P' \quad \{P'\}\ s\ \{Q'\} \quad Q' \Rightarrow Q}{\{P\}\ s\ \{Q\}}$$

Exemple : souvent pratique de partir de la fin :

$$\frac{\{\qquad\qquad\}\ x := x + 1\ \{\qquad\qquad\} \qquad \{\qquad\qquad\}\ x := 2 * x\ \{x > 6\}}{\{\qquad\qquad\}\ x := x + 1; x = 2 * x\ \{x > 6\}}$$

Logique de Hoare (Floyd 1967, Hoare 1969)

Définition (triplet de Hoare) on note $\{P\}\ s\ \{Q\}$ pour :

"Si P est vrai et qu'on lance le programme s, à la fin on aura Q"

Règles de la logique :

$$\frac{}{\{P\}\ \texttt{skip}\ \{P\}} \qquad \frac{}{\{P[e/x]\}\ x := e\ \{P\}}$$

$$\frac{\{P \wedge B\}\ s_1\ \{Q\} \quad \{P \wedge \neg B\}\ s_2\ \{Q\}}{\{P\}\ \texttt{if}\ B\ \texttt{then}\ s_1\ \texttt{else}\ s_2\ \{Q\}} \qquad \frac{\{P\}\ s_1\ \{Q\} \qquad \{Q\}\ s_2\ \{R\}}{\{P\}\ s_1; s_2\ \{R\}}$$

$$\frac{\{I \wedge B\}\ s\ \{I\}}{\{I\}\ \texttt{while}\ B\ \texttt{do}\ s\ \{I \wedge \neg B\}} \qquad \frac{P \Rightarrow P' \quad \{P'\}\ s\ \{Q'\} \quad Q' \Rightarrow Q}{\{P\}\ s\ \{Q\}}$$

Exemple : souvent pratique de partir de la fin :

$$\frac{\{\qquad\qquad\}\ x := x + 1\ \{\qquad\qquad\} \qquad \{2x > 6\}\ x := 2 * x\ \{x > 6\}}{\{\qquad\qquad\}\ x := x + 1; x = 2 * x\ \{x > 6\}}$$

Logique de Hoare (Floyd 1967, Hoare 1969)

Définition (triplet de Hoare) on note $\{P\}\ s\ \{Q\}$ pour :

"Si P est vrai et qu'on lance le programme s, à la fin on aura Q"

Règles de la logique :

$$\frac{}{\{P\}\ \texttt{skip}\ \{P\}} \qquad \frac{}{\{P[e/x]\}\ x := e\ \{P\}}$$

$$\frac{\{P \wedge B\}\ s_1\ \{Q\} \quad \{P \wedge \neg B\}\ s_2\ \{Q\}}{\{P\}\ \texttt{if}\ B\ \texttt{then}\ s_1\ \texttt{else}\ s_2\ \{Q\}} \qquad \frac{\{P\}\ s_1\ \{Q\} \qquad \{Q\}\ s_2\ \{R\}}{\{P\}\ s_1; s_2\ \{R\}}$$

$$\frac{\{I \wedge B\}\ s\ \{I\}}{\{I\}\ \texttt{while}\ B\ \texttt{do}\ s\ \{I \wedge \neg B\}} \qquad \frac{P \Rightarrow P' \quad \{P'\}\ s\ \{Q'\} \quad Q' \Rightarrow Q}{\{P\}\ s\ \{Q\}}$$

Exemple : souvent pratique de partir de la fin :

$$\frac{\{\qquad\qquad\}\ x := x + 1\ \{2x > 6\} \qquad \{2x > 6\}\ x := 2 * x\ \{x > 6\}}{\{\qquad\qquad\}\ x := x + 1; x = 2 * x\ \{x > 6\}}$$

Logique de Hoare (Floyd 1967, Hoare 1969)

Définition (triplet de Hoare) on note $\{P\}\ s\ \{Q\}$ pour :

"Si P est vrai et qu'on lance le programme s, à la fin on aura Q"

Règles de la logique :

$$\frac{}{\{P\}\ \texttt{skip}\ \{P\}} \qquad \frac{}{\{P[e/x]\}\ x := e\ \{P\}}$$

$$\frac{\{P \wedge B\}\ s_1\ \{Q\} \quad \{P \wedge \neg B\}\ s_2\ \{Q\}}{\{P\}\ \texttt{if}\ B\ \texttt{then}\ s_1\ \texttt{else}\ s_2\ \{Q\}} \qquad \frac{\{P\}\ s_1\ \{Q\} \qquad \{Q\}\ s_2\ \{R\}}{\{P\}\ s_1; s_2\ \{R\}}$$

$$\frac{\{I \wedge B\}\ s\ \{I\}}{\{I\}\ \texttt{while}\ B\ \texttt{do}\ s\ \{I \wedge \neg B\}} \qquad \frac{P \Rightarrow P' \quad \{P'\}\ s\ \{Q'\} \quad Q' \Rightarrow Q}{\{P\}\ s\ \{Q\}}$$

Exemple : souvent pratique de partir de la fin :

$$\frac{\{2(x+1) > 6\}\ x := x + 1\ \{2x > 6\} \qquad \{2x > 6\}\ x := 2 * x\ \{x > 6\}}{\{\qquad\qquad\}\ x := x + 1; x = 2 * x\ \{x > 6\}}$$

Logique de Hoare (Floyd 1967, Hoare 1969)

Définition (triplet de Hoare) on note $\{P\}\ s\ \{Q\}$ pour :

"Si P est vrai et qu'on lance le programme s, à la fin on aura Q"

Règles de la logique :

$$\frac{}{\{P\}\ \texttt{skip}\ \{P\}} \qquad \frac{}{\{P[e/x]\}\ x := e\ \{P\}}$$

$$\frac{\{P \wedge B\}\ s_1\ \{Q\} \quad \{P \wedge \neg B\}\ s_2\ \{Q\}}{\{P\}\ \texttt{if}\ B\ \texttt{then}\ s_1\ \texttt{else}\ s_2\ \{Q\}} \qquad \frac{\{P\}\ s_1\ \{Q\} \qquad \{Q\}\ s_2\ \{R\}}{\{P\}\ s_1; s_2\ \{R\}}$$

$$\frac{\{I \wedge B\}\ s\ \{I\}}{\{I\}\ \texttt{while}\ B\ \texttt{do}\ s\ \{I \wedge \neg B\}} \qquad \frac{P \Rightarrow P' \quad \{P'\}\ s\ \{Q'\} \quad Q' \Rightarrow Q}{\{P\}\ s\ \{Q\}}$$

Exemple : souvent pratique de partir de la fin :

$$\frac{\{2(x+1) > 6\}\ x := x + 1\ \{2x > 6\} \qquad \{2x > 6\}\ x := 2 * x\ \{x > 6\}}{\{2(x+1) > 6\}\ x := x + 1; x = 2 * x\ \{x > 6\}}$$

Plus faibles préconditions (Dijkstra, 1975)

Dijkstra a décidé de partir de la fin pour de vrai : il existe une précondition P la plus générale possible telle que $\{P\}\ s\ \{Q\}$.

Plus faibles préconditions (Dijkstra, 1975)

Dijkstra a décidé de partir de la fin pour de vrai : il existe une précondition P la plus générale possible telle que $\{P\}\ s\ \{Q\}$.

Définition de $\mathsf{wp}(s, Q)$ de sorte que $\{\mathsf{wp}(s, Q)\}\ s\ \{Q\}$:

Plus faibles préconditions (Dijkstra, 1975)

Dijkstra a décidé de partir de la fin pour de vrai : il existe une précondition P la plus générale possible telle que $\{P\}\ s\ \{Q\}$.

Définition de $\mathsf{wp}(s, Q)$ de sorte que $\{\mathsf{wp}(s, Q)\}\ s\ \{Q\}$:

$$\mathsf{wp}(\texttt{skip}, Q) \quad = \quad Q$$

Plus faibles préconditions (Dijkstra, 1975)

Dijkstra a décidé de partir de la fin pour de vrai : il existe une précondition P la plus générale possible telle que $\{P\}\ s\ \{Q\}$.

Définition de $\mathsf{wp}(s, Q)$ de sorte que $\{\mathsf{wp}(s, Q)\}\ s\ \{Q\}$:

$$\begin{aligned} \mathsf{wp}(\texttt{skip}, Q) &= Q \\ \mathsf{wp}(x := e, Q) &= Q[e/x] \end{aligned}$$

Plus faibles préconditions (Dijkstra, 1975)

Dijkstra a décidé de partir de la fin pour de vrai : il existe une précondition P la plus générale possible telle que $\{P\}\ s\ \{Q\}$.

Définition de $\mathsf{wp}(s, Q)$ de sorte que $\{\mathsf{wp}(s, Q)\}\ s\ \{Q\}$:

$$\begin{aligned} \mathsf{wp}(\texttt{skip}, Q) &= Q \\ \mathsf{wp}(x := e, Q) &= Q[e/x] \\ \mathsf{wp}(s_1; s_2, Q) &= \mathsf{wp}(s_1, \mathsf{wp}(s_2, Q)) \end{aligned}$$

Plus faibles préconditions (Dijkstra, 1975)

Dijkstra a décidé de partir de la fin pour de vrai : il existe une précondition P la plus générale possible telle que $\{P\}\ s\ \{Q\}$.

Définition de $\mathsf{wp}(s, Q)$ de sorte que $\{\mathsf{wp}(s, Q)\}\ s\ \{Q\}$:

$$
\begin{array}{rcl}
\mathsf{wp}(\texttt{skip}, Q) & = & Q \\
\mathsf{wp}(x := e, Q) & = & Q[e/x] \\
\mathsf{wp}(s_1; s_2, Q) & = & \mathsf{wp}(s_1, \mathsf{wp}(s_2, Q)) \\
\mathsf{wp}(\texttt{if } B \texttt{ then } s_1 \texttt{ else } s_2, Q) & = & (B \Rightarrow \mathsf{wp}(s_1, Q)) \wedge (\neg B \Rightarrow \mathsf{wp}(s_2, Q))
\end{array}
$$

Plus faibles préconditions (Dijkstra, 1975)

Dijkstra a décidé de partir de la fin pour de vrai : il existe une précondition P la plus générale possible telle que $\{P\}\ s\ \{Q\}$.

Définition de $\mathsf{wp}(s, Q)$ de sorte que $\{\mathsf{wp}(s, Q)\}\ s\ \{Q\}$:

$$
\begin{array}{rcl}
\mathsf{wp}(\texttt{skip}, Q) & = & Q \\
\mathsf{wp}(x := e, Q) & = & Q[e/x] \\
\mathsf{wp}(s_1; s_2, Q) & = & \mathsf{wp}(s_1, \mathsf{wp}(s_2, Q)) \\
\mathsf{wp}(\texttt{if } B \texttt{ then } s_1 \texttt{ else } s_2, Q) & = & (B \Rightarrow \mathsf{wp}(s_1, Q)) \wedge (\neg B \Rightarrow \mathsf{wp}(s_2, Q)) \\
\mathsf{wp}(\texttt{while } B \texttt{ do } s, Q) & = & \exists I \quad I \wedge (B \wedge I \Rightarrow \mathsf{wp}(s, I)) \\
 & & \qquad \wedge (\neg B \wedge I \Rightarrow Q)
\end{array}
$$

Plus faibles préconditions (Dijkstra, 1975)

Dijkstra a décidé de partir de la fin pour de vrai : il existe une précondition P la plus générale possible telle que $\{P\}\ s\ \{Q\}$.

Définition de $\mathsf{wp}(s, Q)$ de sorte que $\{\mathsf{wp}(s, Q)\}\ s\ \{Q\}$:

$$
\begin{array}{rcl}
\mathsf{wp}(\texttt{skip}, Q) & = & Q \\
\mathsf{wp}(x := e, Q) & = & Q[e/x] \\
\mathsf{wp}(s_1; s_2, Q) & = & \mathsf{wp}(s_1, \mathsf{wp}(s_2, Q)) \\
\mathsf{wp}(\texttt{if } B \texttt{ then } s_1 \texttt{ else } s_2, Q) & = & (B \Rightarrow \mathsf{wp}(s_1, Q)) \wedge (\neg B \Rightarrow \mathsf{wp}(s_2, Q)) \\
\mathsf{wp}(\texttt{while } B \texttt{ do } s, Q) & = & \exists I \quad I \wedge (B \wedge I \Rightarrow \mathsf{wp}(s, I)) \\
 & & \qquad \wedge (\neg B \wedge I \Rightarrow Q)
\end{array}
$$

Exemples :

- $\mathsf{wp}(x := 4,\ x \geqslant 1) = (4 \geqslant 1)$

Plus faibles préconditions (Dijkstra, 1975)

Dijkstra a décidé de partir de la fin pour de vrai : il existe une précondition P la plus générale possible telle que $\{P\}\ s\ \{Q\}$.

Définition de $\mathsf{wp}(s, Q)$ de sorte que $\{\mathsf{wp}(s, Q)\}\ s\ \{Q\}$:

$$
\begin{aligned}
\mathsf{wp}(\texttt{skip}, Q) &= Q \\
\mathsf{wp}(x := e, Q) &= Q[e/x] \\
\mathsf{wp}(s_1; s_2, Q) &= \mathsf{wp}(s_1, \mathsf{wp}(s_2, Q)) \\
\mathsf{wp}(\texttt{if } B \texttt{ then } s_1 \texttt{ else } s_2, Q) &= (B \Rightarrow \mathsf{wp}(s_1, Q)) \wedge (\neg B \Rightarrow \mathsf{wp}(s_2, Q)) \\
\mathsf{wp}(\texttt{while } B \texttt{ do } s, Q) &= \exists I \quad I \wedge (B \wedge I \Rightarrow \mathsf{wp}(s, I)) \\
&\qquad\qquad \wedge (\neg B \wedge I \Rightarrow Q)
\end{aligned}
$$

Exemples :

- $\mathsf{wp}(x := 4,\ x \geqslant 1) = (4 \geqslant 1)$
- $\mathsf{wp}((\texttt{if } x > 0 \texttt{ then } r := x \texttt{ else } r := 0 - x),\ r = |x|) =$
  $(x > 0 \Rightarrow x = |x|) \wedge (x \leqslant 0 \Rightarrow (0 - x = |x|))$

Exemple : preuve du calcul de x^n

Programme avec postcondition $r = x^n$:

$$
\begin{array}{l}
i := n; \\
r := 1; \\
\texttt{while } i > 0 \texttt{ do} \\
\quad (r := r * x; \\
\quad\ i := i - 1)
\end{array}
$$

On est fier de trouver un bel invariant $I \ = \ (rx^i = x^n)$

Exemple : preuve du calcul de x^n

Programme avec postcondition $r = x^n$:

$$
\begin{array}{l}
i := n; \\
r := 1; \\
\texttt{while } i > 0 \texttt{ do} \\
\quad (r := r * x; \\
\quad\ i := i - 1)
\end{array}
$$

On est fier de trouver un bel invariant $I \ = \ (rx^i = x^n)$

$\mathsf{wp}(i := n; r := 1; \texttt{while } i > 0 \texttt{ do } (r := r * x; i := i - 1))(r = x^n)$

Exemple : preuve du calcul de x^n

Programme avec postcondition $r = x^n$:

$$
\begin{array}{l}
i := n; \\
r := 1; \\
\texttt{while } i > 0 \texttt{ do} \\
\quad (r := r * x; \\
\quad\ i := i - 1)
\end{array}
$$

On est fier de trouver un bel invariant $I \ = \ (rx^i = x^n)$

$$
\begin{aligned}
&\mathsf{wp}(i := n; r := 1; \texttt{while } i > 0 \texttt{ do } (r := r * x; i := i - 1))(r = x^n) \\
&= \mathsf{wp}(r := 1; \texttt{while } i > 0 \texttt{ do } (r := r * x; i := i - 1))(r = x^n)[n/i]
\end{aligned}
$$

Exemple : preuve du calcul de x^n

Programme avec postcondition $r = x^n$:

$$
\begin{array}{l}
i := n; \\
r := 1; \\
\texttt{while } i > 0 \texttt{ do} \\
\quad (r := r * x; \\
\quad\ i := i - 1)
\end{array}
$$

On est fier de trouver un bel invariant $I \ = \ (rx^i = x^n)$

$$
\begin{aligned}
&\mathsf{wp}(i := n; r := 1; \texttt{while } i > 0 \texttt{ do } (r := r * x; i := i - 1))(r = x^n) \\
&= \mathsf{wp}(r := 1; \texttt{while } i > 0 \texttt{ do } (r := r * x; i := i - 1))(r = x^n)[n/i] \\
&= \mathsf{wp}(\texttt{while } i > 0 \texttt{ do } (r := r * x; i := i - 1))(r = x^n)[n/i][1/r]
\end{aligned}
$$

Exemple : preuve du calcul de x^n

Programme avec postcondition $r = x^n$:

$$
\begin{array}{l}
i := n; \\
r := 1; \\
\texttt{while } i > 0 \texttt{ do} \\
\quad (r := r * x; \\
\quad\ i := i - 1)
\end{array}
$$

On est fier de trouver un bel invariant $I \ = \ (rx^i = x^n)$

$$
\begin{aligned}
&\mathsf{wp}(i := n; r := 1; \texttt{while } i > 0 \texttt{ do } (r := r * x; i := i - 1))(r = x^n) \\
&= \mathsf{wp}(r := 1; \texttt{while } i > 0 \texttt{ do } (r := r * x; i := i - 1))(r = x^n)[n/i] \\
&= \mathsf{wp}(\texttt{while } i > 0 \texttt{ do } (r := r * x; i := i - 1))(r = x^n)[n/i][1/r] \\
&= I[n/i][1/r] \wedge (i > 0 \wedge I \Rightarrow \mathsf{wp}(r := r * x; i := i - 1, I)) \wedge (i \leqslant 0 \wedge I \Rightarrow r = x^n)
\end{aligned}
$$

Exemple : preuve du calcul de x^n

Programme avec postcondition $r = x^n$:

$$
\begin{array}{l}
i := n; \\
r := 1; \\
\texttt{while } i > 0 \texttt{ do} \\
\quad (r := r * x; \\
\quad\ i := i - 1)
\end{array}
$$

On est fier de trouver un bel invariant $I \ = \ (rx^i = x^n)$

$$
\begin{aligned}
&\mathsf{wp}(i := n; r := 1; \texttt{while } i > 0 \texttt{ do } (r := r * x; i := i - 1))(r = x^n) \\
&= \mathsf{wp}(r := 1; \texttt{while } i > 0 \texttt{ do } (r := r * x; i := i - 1))(r = x^n)[n/i] \\
&= \mathsf{wp}(\texttt{while } i > 0 \texttt{ do } (r := r * x; i := i - 1))(r = x^n)[n/i][1/r] \\
&= I[n/i][1/r] \wedge (i > 0 \wedge I \Rightarrow \mathsf{wp}(r := r * x; i := i - 1, I)) \wedge (i \leqslant 0 \wedge I \Rightarrow r = x^n) \\
&= I[n/i][1/r] \wedge (i > 0 \wedge I \Rightarrow \mathsf{wp}(r := r * x, \mathsf{wp}(i := i - 1, I))) \wedge (i \leqslant 0 \wedge I \Rightarrow r = x^n)
\end{aligned}
$$

Exemple : preuve du calcul de x^n

Programme avec postcondition $r = x^n$:

$$
\begin{array}{l}
i := n; \\
r := 1; \\
\texttt{while } i > 0 \texttt{ do} \\
\quad (r := r * x; \\
\quad\ i := i - 1)
\end{array}
$$

On est fier de trouver un bel invariant $I \ = \ (rx^i = x^n)$

$$
\begin{aligned}
&\mathsf{wp}(i := n; r := 1; \texttt{while } i > 0 \texttt{ do } (r := r * x; i := i - 1))(r = x^n) \\
&= \mathsf{wp}(r := 1; \texttt{while } i > 0 \texttt{ do } (r := r * x; i := i - 1))(r = x^n)[n/i] \\
&= \mathsf{wp}(\texttt{while } i > 0 \texttt{ do } (r := r * x; i := i - 1))(r = x^n)[n/i][1/r] \\
&= I[n/i][1/r] \wedge (i > 0 \wedge I \Rightarrow \mathsf{wp}(r := r * x; i := i - 1, I)) \wedge (i \leqslant 0 \wedge I \Rightarrow r = x^n) \\
&= I[n/i][1/r] \wedge (i > 0 \wedge I \Rightarrow \mathsf{wp}(r := r * x, \mathsf{wp}(i := i - 1, I))) \wedge (i \leqslant 0 \wedge I \Rightarrow r = x^n) \\
&= I[n/i][1/r] \wedge (i > 0 \wedge I \Rightarrow \mathsf{wp}(r := r * x, I[(i-1)/i])) \wedge (i \leqslant 0 \wedge I \Rightarrow r = x^n)
\end{aligned}
$$

Exemple : preuve du calcul de x^n

Programme avec postcondition $r = x^n$:

$$
\begin{array}{l}
i := n; \\
r := 1; \\
\texttt{while } i > 0 \texttt{ do} \\
\quad (r := r * x; \\
\quad\ i := i - 1)
\end{array}
$$

On est fier de trouver un bel invariant $I \ = \ (rx^i = x^n)$

$$
\begin{aligned}
&\mathsf{wp}(i := n; r := 1; \texttt{while } i > 0 \texttt{ do } (r := r * x; i := i - 1))(r = x^n) \\
&= \mathsf{wp}(r := 1; \texttt{while } i > 0 \texttt{ do } (r := r * x; i := i - 1))(r = x^n)[n/i] \\
&= \mathsf{wp}(\texttt{while } i > 0 \texttt{ do } (r := r * x; i := i - 1))(r = x^n)[n/i][1/r] \\
&= I[n/i][1/r] \wedge (i > 0 \wedge I \Rightarrow \mathsf{wp}(r := r * x; i := i - 1, I)) \wedge (i \leqslant 0 \wedge I \Rightarrow r = x^n) \\
&= I[n/i][1/r] \wedge (i > 0 \wedge I \Rightarrow \mathsf{wp}(r := r * x, \mathsf{wp}(i := i - 1, I))) \wedge (i \leqslant 0 \wedge I \Rightarrow r = x^n) \\
&= I[n/i][1/r] \wedge (i > 0 \wedge I \Rightarrow \mathsf{wp}(r := r * x, I[(i-1)/i])) \wedge (i \leqslant 0 \wedge I \Rightarrow r = x^n) \\
&= I[n/i][1/r] \wedge (i > 0 \wedge I \Rightarrow I[(i-1)/i][(rx)/x]) \wedge (i \leqslant 0 \wedge I \Rightarrow r = x^n)
\end{aligned}
$$

Exemple : preuve du calcul de x^n

Programme avec postcondition $r = x^n$:

$$
\begin{array}{l}
i := n; \\
r := 1; \\
\texttt{while } i > 0 \texttt{ do} \\
\quad (r := r * x; \\
\quad\ i := i - 1)
\end{array}
$$

On est fier de trouver un bel invariant $I \ = \ (rx^i = x^n)$

$$
\begin{aligned}
&\mathsf{wp}(i := n; r := 1; \texttt{while } i > 0 \texttt{ do } (r := r * x; i := i - 1))(r = x^n) \\
&= \mathsf{wp}(r := 1; \texttt{while } i > 0 \texttt{ do } (r := r * x; i := i - 1))(r = x^n)[n/i] \\
&= \mathsf{wp}(\texttt{while } i > 0 \texttt{ do } (r := r * x; i := i - 1))(r = x^n)[n/i][1/r] \\
&= I[n/i][1/r] \wedge (i > 0 \wedge I \Rightarrow \mathsf{wp}(r := r * x; i := i - 1, I)) \wedge (i \leqslant 0 \wedge I \Rightarrow r = x^n) \\
&= I[n/i][1/r] \wedge (i > 0 \wedge I \Rightarrow \mathsf{wp}(r := r * x, \mathsf{wp}(i := i - 1, I))) \wedge (i \leqslant 0 \wedge I \Rightarrow r = x^n) \\
&= I[n/i][1/r] \wedge (i > 0 \wedge I \Rightarrow \mathsf{wp}(r := r * x, I[(i-1)/i])) \wedge (i \leqslant 0 \wedge I \Rightarrow r = x^n) \\
&= I[n/i][1/r] \wedge (i > 0 \wedge I \Rightarrow I[(i-1)/i][(rx)/x]) \wedge (i \leqslant 0 \wedge I \Rightarrow r = x^n) \\
&= 1x^n = x^n \wedge (i > 0 \wedge rx^i = x^n \Rightarrow rx \cdot x^{i-1} = x^n) \wedge (i \leqslant 0 \wedge rx^i = x^n \Rightarrow r = x^n)
\end{aligned}
$$

Exemple : preuve du calcul de x^n

Programme avec postcondition $r = x^n$:

$$
\begin{array}{l}
i := n; \\
r := 1; \\
\texttt{while } i > 0 \texttt{ do} \\
\quad (r := r * x; \\
\quad\ i := i - 1)
\end{array}
$$

On est fier de trouver un bel invariant $I \ = \ (rx^i = x^n)$

$$
\begin{aligned}
&\mathsf{wp}(i := n; r := 1; \texttt{while } i > 0 \texttt{ do } (r := r * x; i := i - 1))(r = x^n) \\
&= \mathsf{wp}(r := 1; \texttt{while } i > 0 \texttt{ do } (r := r * x; i := i - 1))(r = x^n)[n/i] \\
&= \mathsf{wp}(\texttt{while } i > 0 \texttt{ do } (r := r * x; i := i - 1))(r = x^n)[n/i][1/r] \\
&= I[n/i][1/r] \wedge (i > 0 \wedge I \Rightarrow \mathsf{wp}(r := r * x; i := i - 1, I)) \wedge (i \leqslant 0 \wedge I \Rightarrow r = x^n) \\
&= I[n/i][1/r] \wedge (i > 0 \wedge I \Rightarrow \mathsf{wp}(r := r * x, \mathsf{wp}(i := i - 1, I))) \wedge (i \leqslant 0 \wedge I \Rightarrow r = x^n) \\
&= I[n/i][1/r] \wedge (i > 0 \wedge I \Rightarrow \mathsf{wp}(r := r * x, I[(i-1)/i])) \wedge (i \leqslant 0 \wedge I \Rightarrow r = x^n) \\
&= I[n/i][1/r] \wedge (i > 0 \wedge I \Rightarrow I[(i-1)/i][(rx)/x]) \wedge (i \leqslant 0 \wedge I \Rightarrow r = x^n) \\
&= 1x^n = x^n \wedge (i > 0 \wedge rx^i = x^n \Rightarrow rx \cdot x^{i-1} = x^n) \wedge (i \leqslant 0 \wedge rx^i = x^n \Rightarrow r = x^n) \\
&= x^n = x^n \wedge (i > 0 \wedge rx^i = x^n \Rightarrow rx^i = x^n) \wedge (i \leqslant 0 \wedge rx^i = x^n \Rightarrow r = x^n) = \text{Faux?}
\end{aligned}
$$

Exemple : preuve du calcul de x^n

Programme avec postcondition $r = x^n$:

$$
\begin{array}{l}
i := n; \\
r := 1; \\
\texttt{while } i > 0 \texttt{ do} \\
\quad (r := r * x; \\
\quad\ i := i - 1)
\end{array}
$$

On est fier de trouver un bel invariant $I \ = \ (rx^i = x^n \wedge i \geqslant 0)$

$$
\begin{aligned}
&\mathsf{wp}(i := n; r := 1; \texttt{while } i > 0 \texttt{ do } (r := r * x; i := i - 1))(r = x^n) \\
&= \mathsf{wp}(r := 1; \texttt{while } i > 0 \texttt{ do } (r := r * x; i := i - 1))(r = x^n)[n/i] \\
&= \mathsf{wp}(\texttt{while } i > 0 \texttt{ do } (r := r * x; i := i - 1))(r = x^n)[n/i][1/r] \\
&= I[n/i][1/r] \wedge (i > 0 \wedge I \Rightarrow \mathsf{wp}(r := r * x; i := i - 1, I)) \wedge (i \leqslant 0 \wedge I \Rightarrow r = x^n) \\
&= I[n/i][1/r] \wedge (i > 0 \wedge I \Rightarrow \mathsf{wp}(r := r * x, \mathsf{wp}(i := i - 1, I))) \wedge (i \leqslant 0 \wedge I \Rightarrow r = x^n) \\
&= I[n/i][1/r] \wedge (i > 0 \wedge I \Rightarrow \mathsf{wp}(r := r * x, I[(i-1)/i])) \wedge (i \leqslant 0 \wedge I \Rightarrow r = x^n) \\
&= I[n/i][1/r] \wedge (i > 0 \wedge I \Rightarrow I[(i-1)/i][(rx)/x]) \wedge (i \leqslant 0 \wedge I \Rightarrow r = x^n) \\
&= 1x^n = x^n \wedge n \geqslant 0 \wedge (i > 0 \wedge rx^i = x^n \wedge i \geqslant 0 \Rightarrow rx \cdot x^{i-1} = x^n \wedge i - 1 \geqslant 0) \\
&\qquad\qquad \wedge (i \leqslant 0 \wedge rx^i = x^n \wedge i \geqslant 0 \Rightarrow r = x^n)
\end{aligned}
$$

Exemple : preuve du calcul de x^n

Programme avec postcondition $r = x^n$:

$$
\begin{array}{l}
i := n; \\
r := 1; \\
\texttt{while } i > 0 \texttt{ do} \\
\quad (r := r * x; \\
\quad\ i := i - 1)
\end{array}
$$

On est fier de trouver un bel invariant $I \ = \ (rx^i = x^n \wedge i \geqslant 0)$

$$
\begin{aligned}
&\mathsf{wp}(i := n; r := 1; \texttt{while } i > 0 \texttt{ do } (r := r * x; i := i - 1))(r = x^n) \\
&= \mathsf{wp}(r := 1; \texttt{while } i > 0 \texttt{ do } (r := r * x; i := i - 1))(r = x^n)[n/i] \\
&= \mathsf{wp}(\texttt{while } i > 0 \texttt{ do } (r := r * x; i := i - 1))(r = x^n)[n/i][1/r] \\
&= I[n/i][1/r] \wedge (i > 0 \wedge I \Rightarrow \mathsf{wp}(r := r * x; i := i - 1, I)) \wedge (i \leqslant 0 \wedge I \Rightarrow r = x^n) \\
&= I[n/i][1/r] \wedge (i > 0 \wedge I \Rightarrow \mathsf{wp}(r := r * x, \mathsf{wp}(i := i - 1, I))) \wedge (i \leqslant 0 \wedge I \Rightarrow r = x^n) \\
&= I[n/i][1/r] \wedge (i > 0 \wedge I \Rightarrow \mathsf{wp}(r := r * x, I[(i-1)/i])) \wedge (i \leqslant 0 \wedge I \Rightarrow r = x^n) \\
&= I[n/i][1/r] \wedge (i > 0 \wedge I \Rightarrow I[(i-1)/i][(rx)/x]) \wedge (i \leqslant 0 \wedge I \Rightarrow r = x^n) \\
&= 1x^n = x^n \wedge n \geqslant 0 \wedge (i > 0 \wedge rx^i = x^n \wedge i \geqslant 0 \Rightarrow rx \cdot x^{i-1} = x^n \wedge i - 1 \geqslant 0) \\
&\qquad\qquad \wedge (i \leqslant 0 \wedge rx^i = x^n \wedge i \geqslant 0 \Rightarrow r = x^n) \\
&= n \geqslant 0
\end{aligned}
$$

Exemple : preuve du calcul de x^n

Programme avec postcondition $r = x^n$:

$$
\begin{array}{l}
i := n; \\
r := 1; \\
\texttt{while } i > 0 \texttt{ do} \\
\quad (r := r * x; \\
\quad\ i := i - 1)
\end{array}
$$

On est fier de trouver un bel invariant $I \ = \ (rx^i = x^n \wedge i \geqslant 0)$

$$
\begin{aligned}
&\mathsf{wp}(i := n; r := 1; \texttt{while } i > 0 \texttt{ do } (r := r * x; i := i - 1))(r = x^n) \\
&= \mathsf{wp}(r := 1; \texttt{while } i > 0 \texttt{ do } (r := r * x; i := i - 1))(r = x^n)[n/i] \\
&= \mathsf{wp}(\texttt{while } i > 0 \texttt{ do } (r := r * x; i := i - 1))(r = x^n)[n/i][1/r] \\
&= I[n/i][1/r] \wedge (i > 0 \wedge I \Rightarrow \mathsf{wp}(r := r * x; i := i - 1, I)) \wedge (i \leqslant 0 \wedge I \Rightarrow r = x^n) \\
&= I[n/i][1/r] \wedge (i > 0 \wedge I \Rightarrow \mathsf{wp}(r := r * x, \mathsf{wp}(i := i - 1, I))) \wedge (i \leqslant 0 \wedge I \Rightarrow r = x^n) \\
&= I[n/i][1/r] \wedge (i > 0 \wedge I \Rightarrow \mathsf{wp}(r := r * x, I[(i-1)/i])) \wedge (i \leqslant 0 \wedge I \Rightarrow r = x^n) \\
&= I[n/i][1/r] \wedge (i > 0 \wedge I \Rightarrow I[(i-1)/i][(rx)/x]) \wedge (i \leqslant 0 \wedge I \Rightarrow r = x^n) \\
&= 1x^n = x^n \wedge n \geqslant 0 \wedge (i > 0 \wedge rx^i = x^n \wedge i \geqslant 0 \Rightarrow rx \cdot x^{i-1} = x^n \wedge i - 1 \geqslant 0) \\
&\qquad\qquad \wedge (i \leqslant 0 \wedge rx^i = x^n \wedge i \geqslant 0 \Rightarrow r = x^n) \\
&= n \geqslant 0
\end{aligned}
$$

Typique : invariant pas assez général, oubli de précondition

Fini ?

Presque... on doit montrer qu'on a les bonnes règles.

Qu'est-ce que ça veut dire ?

Prouver les règles ? Correction de la logique de Hoare

Si avec les règles on montre $\{P\}\ s\ \{Q\}$ alors :
"Si P **est vrai** et qu'on lance le programme s, à la fin **on aura** Q"

Prouver les règles ? Correction de la logique de Hoare

Si avec les règles on montre $\{P\}\ s\ \{Q\}$ alors :
"Si P **est vrai** et qu'on lance le programme s, à la fin **on aura** Q"

P et Q parlent de la mémoire, qu'on modélise donc :

- la mémoire est une fonction partielle m : variables $\rightharpoonup \mathbb{Z}$

Prouver les règles ? Correction de la logique de Hoare

Si avec les règles on montre $\{P\}\ s\ \{Q\}$ alors :
"Si P **est vrai** et qu'on lance le programme s, à la fin **on aura** Q"

P et Q parlent de la mémoire, qu'on modélise donc :

- la mémoire est une fonction partielle m : variables $\rightharpoonup \mathbb{Z}$
- étendue aux expressions : m : expressions $\rightharpoonup \mathbb{Z}$

Prouver les règles ? Correction de la logique de Hoare

Si avec les règles on montre $\{P\}\ s\ \{Q\}$ alors :
"Si P **est vrai** et qu'on lance le programme s, à la fin **on aura** Q"

P et Q parlent de la mémoire, qu'on modélise donc :

- la mémoire est une fonction partielle m : variables $\rightharpoonup \mathbb{Z}$
- étendue aux expressions : m : expressions $\rightharpoonup \mathbb{Z}$
- les assertions P et Q sont des prédicats sur la mémoire.

Prouver les règles ? Correction de la logique de Hoare

Si avec les règles on montre $\{P\}\ s\ \{Q\}$ alors :
"Si P **est vrai** et qu'on lance le programme s, à la fin **on aura** Q"

P et Q parlent de la mémoire, qu'on modélise donc :

- la mémoire est une fonction partielle m : variables $\rightharpoonup \mathbb{Z}$
- étendue aux expressions : m : expressions $\rightharpoonup \mathbb{Z}$
- les assertions P et Q sont des prédicats sur la mémoire.

Exemples avec $m = (x \mapsto 3, y \mapsto 4)$:

- $m(x + 2y) = 11$

Prouver les règles ? Correction de la logique de Hoare

Si avec les règles on montre $\{P\}\ s\ \{Q\}$ alors :
"Si P **est vrai** et qu'on lance le programme s, à la fin **on aura** Q"

P et Q parlent de la mémoire, qu'on modélise donc :

- la mémoire est une fonction partielle m : variables $\rightharpoonup \mathbb{Z}$
- étendue aux expressions : m : expressions $\rightharpoonup \mathbb{Z}$
- les assertions P et Q sont des prédicats sur la mémoire.

Exemples avec $m = (x \mapsto 3, y \mapsto 4)$:

- $m(x + 2y) = 11$
- si $P = (x \geqslant 2 \vee x = 0)$ on a bien $P(m)$

Prouver les règles ? Correction de la logique de Hoare

"Si P est vrai et qu'on **lance le programme** s, à la fin on aura Q"

Prouver les règles ? Correction de la logique de Hoare

"Si P est vrai et qu'on **lance le programme** s, à la fin on aura Q"

On modélise l'exécution par une **sémantique opérationnelle**, c'est à dire une relation de réduction $m, s \rightarrow m', s'$

Prouver les règles ? Correction de la logique de Hoare

"Si P est vrai et qu'on **lance le programme** s, à la fin on aura Q"

On modélise l'exécution par une **sémantique opérationnelle**, c'est à dire une relation de réduction $m, s \;\rightarrow\; m', s'$

$$\frac{}{m, x := e \;\rightarrow\; m(x \mapsto m(e)), \texttt{skip}}$$

Prouver les règles ? Correction de la logique de Hoare

"Si P est vrai et qu'on **lance le programme** s, à la fin on aura Q"

On modélise l'exécution par une **sémantique opérationnelle**, c'est à dire une relation de réduction $m, s \;\rightarrow\; m', s'$

$$\frac{}{m, x := e \;\rightarrow\; m(x \mapsto m(e)), \texttt{skip}}$$

$$\frac{}{m, (\texttt{skip}; s) \;\rightarrow\; m, s}$$

Prouver les règles ? Correction de la logique de Hoare

"Si P est vrai et qu'on **lance le programme** s, à la fin on aura Q"

On modélise l'exécution par une **sémantique opérationnelle**, c'est à dire une relation de réduction $m, s \;\rightarrow\; m', s'$

$$\frac{}{m, x := e \;\rightarrow\; m(x \mapsto m(e)), \texttt{skip}}$$

$$\frac{}{m, (\texttt{skip}; s) \;\rightarrow\; m, s} \qquad \frac{m, s_1 \;\rightarrow\; m', s_1'}{m, (s_1; s_2) \;\rightarrow\; m', (s_1'; s_2)}$$

Prouver les règles ? Correction de la logique de Hoare

"Si P est vrai et qu'on **lance le programme** s, à la fin on aura Q"

On modélise l'exécution par une **sémantique opérationnelle**, c'est à dire une relation de réduction $m, s \;\to\; m', s'$

$$\frac{}{m, x := e \;\to\; m(x \mapsto m(e)), \texttt{skip}}$$

$$\frac{}{m, (\texttt{skip}; s) \;\to\; m, s} \qquad \frac{m, s_1 \;\to\; m', s_1'}{m, (s_1; s_2) \;\to\; m', (s_1'; s_2)}$$

$$\frac{m(e) \neq 0}{m, \texttt{if } e \texttt{ then } s_1 \texttt{ else } s_2 \;\to\; m, s_1} \qquad \frac{m(e) = 0}{m, \texttt{if } e \texttt{ then } s_1 \texttt{ else } s_2 \;\to\; m, s_2}$$

Prouver les règles ? Correction de la logique de Hoare

"Si P est vrai et qu'on **lance le programme** s, à la fin on aura Q"

On modélise l'exécution par une **sémantique opérationnelle**, c'est à dire une relation de réduction $m, s \;\to\; m', s'$

$$\frac{}{m, x := e \;\to\; m(x \mapsto m(e)), \texttt{skip}}$$

$$\frac{}{m, (\texttt{skip}; s) \;\to\; m, s} \qquad \frac{m, s_1 \;\to\; m', s_1'}{m, (s_1; s_2) \;\to\; m', (s_1'; s_2)}$$

$$\frac{m(e) \neq 0}{m, \texttt{if } e \texttt{ then } s_1 \texttt{ else } s_2 \;\to\; m, s_1} \qquad \frac{m(e) = 0}{m, \texttt{if } e \texttt{ then } s_1 \texttt{ else } s_2 \;\to\; m, s_2}$$

$$\frac{m(e) \neq 0}{m, \texttt{while } e \texttt{ do } s \;\to\; m, (s; \texttt{while } e \texttt{ do } s)} \qquad \frac{m(e) = 0}{m, \texttt{while } e \texttt{ do } s \;\to\; m, \texttt{skip}}$$

Sémantique opérationnelle : exemple

$$(n \mapsto 5), (i := n; r := 1; \texttt{while}\ i > 0\ \texttt{do}\ (r := r * 2; i := i - 1))$$

Sémantique opérationnelle : exemple

$$
\begin{array}{ll}
 & (n \mapsto 5), (i := n; r := 1; \texttt{while}\ i > 0\ \texttt{do}\ (r := r * 2; i := i - 1)) \\
\rightarrow & (n \mapsto 5, i \mapsto 5), (r := 1; \texttt{while}\ i > 0\ \texttt{do}\ (r := r * 2; i := i - 1))
\end{array}
$$

Sémantique opérationnelle : exemple

$$
\begin{array}{ll}
 & (n \mapsto 5), (i := n; r := 1; \texttt{while}\ i > 0\ \texttt{do}\ (r := r * 2; i := i - 1)) \\
\rightarrow & (n \mapsto 5, i \mapsto 5), (r := 1; \texttt{while}\ i > 0\ \texttt{do}\ (r := r * 2; i := i - 1)) \\
\rightarrow & (n \mapsto 5, i \mapsto 5, r \mapsto 1), (\texttt{while}\ i > 0\ \texttt{do}\ (r := r * 2; i := i - 1))
\end{array}
$$

Sémantique opérationnelle : exemple

$$
\begin{array}{ll}
 & (n \mapsto 5), (i := n; r := 1; \texttt{while}\ i > 0\ \texttt{do}\ (r := r * 2; i := i - 1)) \\
\rightarrow & (n \mapsto 5, i \mapsto 5), (r := 1; \texttt{while}\ i > 0\ \texttt{do}\ (r := r * 2; i := i - 1)) \\
\rightarrow & (n \mapsto 5, i \mapsto 5, r \mapsto 1), (\texttt{while}\ i > 0\ \texttt{do}\ (r := r * 2; i := i - 1)) \\
\rightarrow & (n \mapsto 5, i \mapsto 5, r \mapsto 1), (r := r * 2; i := i - 1; \texttt{while}\ i > 0\ \texttt{do}\ \ldots)
\end{array}
$$

Sémantique opérationnelle : exemple

$$
\begin{array}{ll}
 & (n \mapsto 5), (i := n; r := 1; \texttt{while}\ i > 0\ \texttt{do}\ (r := r * 2; i := i - 1)) \\
\rightarrow & (n \mapsto 5, i \mapsto 5), (r := 1; \texttt{while}\ i > 0\ \texttt{do}\ (r := r * 2; i := i - 1)) \\
\rightarrow & (n \mapsto 5, i \mapsto 5, r \mapsto 1), (\texttt{while}\ i > 0\ \texttt{do}\ (r := r * 2; i := i - 1)) \\
\rightarrow & (n \mapsto 5, i \mapsto 5, r \mapsto 1), (r := r * 2; i := i - 1; \texttt{while}\ i > 0\ \texttt{do}\ \ldots) \\
\rightarrow & (n \mapsto 5, i \mapsto 5, r \mapsto 2), (i := i - 1; \texttt{while}\ i > 0\ \texttt{do}\ \ldots)
\end{array}
$$

Sémantique opérationnelle : exemple

$$
\begin{array}{ll}
 & (n \mapsto 5), (i := n; r := 1; \texttt{while}\ i > 0\ \texttt{do}\ (r := r * 2; i := i - 1)) \\
\rightarrow & (n \mapsto 5, i \mapsto 5), (r := 1; \texttt{while}\ i > 0\ \texttt{do}\ (r := r * 2; i := i - 1)) \\
\rightarrow & (n \mapsto 5, i \mapsto 5, r \mapsto 1), (\texttt{while}\ i > 0\ \texttt{do}\ (r := r * 2; i := i - 1)) \\
\rightarrow & (n \mapsto 5, i \mapsto 5, r \mapsto 1), (r := r * 2; i := i - 1; \texttt{while}\ i > 0\ \texttt{do}\ \ldots) \\
\rightarrow & (n \mapsto 5, i \mapsto 5, r \mapsto 2), (i := i - 1; \texttt{while}\ i > 0\ \texttt{do}\ \ldots) \\
\rightarrow & (n \mapsto 5, i \mapsto 4, r \mapsto 2), (\texttt{while}\ i > 0\ \texttt{do}\ \ldots)
\end{array}
$$

Sémantique opérationnelle : exemple

$$
\begin{array}{rl}
 & (n \mapsto 5), (i := n; r := 1; \texttt{while}\ i > 0\ \texttt{do}\ (r := r * 2; i := i - 1)) \\
\rightarrow & (n \mapsto 5, i \mapsto 5), (r := 1; \texttt{while}\ i > 0\ \texttt{do}\ (r := r * 2; i := i - 1)) \\
\rightarrow & (n \mapsto 5, i \mapsto 5, r \mapsto 1), (\texttt{while}\ i > 0\ \texttt{do}\ (r := r * 2; i := i - 1)) \\
\rightarrow & (n \mapsto 5, i \mapsto 5, r \mapsto 1), (r := r * 2; i := i - 1; \texttt{while}\ i > 0\ \texttt{do}\ \ldots) \\
\rightarrow & (n \mapsto 5, i \mapsto 5, r \mapsto 2), (i := i - 1; \texttt{while}\ i > 0\ \texttt{do}\ \ldots) \\
\rightarrow & (n \mapsto 5, i \mapsto 4, r \mapsto 2), (\texttt{while}\ i > 0\ \texttt{do}\ \ldots) \\
\rightarrow\rightarrow\rightarrow & (n \mapsto 5, i \mapsto 3, r \mapsto 4), (\texttt{while}\ i > 0\ \texttt{do}\ (r := r * 2; i := i - 1))
\end{array}
$$

Sémantique opérationnelle : exemple

$$
\begin{array}{rl}
 & (n \mapsto 5), (i := n; r := 1; \texttt{while}\ i > 0\ \texttt{do}\ (r := r * 2; i := i - 1)) \\
\rightarrow & (n \mapsto 5, i \mapsto 5), (r := 1; \texttt{while}\ i > 0\ \texttt{do}\ (r := r * 2; i := i - 1)) \\
\rightarrow & (n \mapsto 5, i \mapsto 5, r \mapsto 1), (\texttt{while}\ i > 0\ \texttt{do}\ (r := r * 2; i := i - 1)) \\
\rightarrow & (n \mapsto 5, i \mapsto 5, r \mapsto 1), (r := r * 2; i := i - 1; \texttt{while}\ i > 0\ \texttt{do}\ \ldots) \\
\rightarrow & (n \mapsto 5, i \mapsto 5, r \mapsto 2), (i := i - 1; \texttt{while}\ i > 0\ \texttt{do}\ \ldots) \\
\rightarrow & (n \mapsto 5, i \mapsto 4, r \mapsto 2), (\texttt{while}\ i > 0\ \texttt{do}\ \ldots) \\
\rightarrow\rightarrow\rightarrow & (n \mapsto 5, i \mapsto 3, r \mapsto 4), (\texttt{while}\ i > 0\ \texttt{do}\ (r := r * 2; i := i - 1)) \\
\rightarrow\rightarrow\rightarrow & (n \mapsto 5, i \mapsto 2, r \mapsto 8), (\texttt{while}\ i > 0\ \texttt{do}\ (r := r * 2; i := i - 1))
\end{array}
$$

Sémantique opérationnelle : exemple

$$
\begin{array}{ll}
 & (n \mapsto 5), (i := n; r := 1; \texttt{while}\ i > 0\ \texttt{do}\ (r := r * 2; i := i - 1)) \\
\rightarrow & (n \mapsto 5, i \mapsto 5), (r := 1; \texttt{while}\ i > 0\ \texttt{do}\ (r := r * 2; i := i - 1)) \\
\rightarrow & (n \mapsto 5, i \mapsto 5, r \mapsto 1), (\texttt{while}\ i > 0\ \texttt{do}\ (r := r * 2; i := i - 1)) \\
\rightarrow & (n \mapsto 5, i \mapsto 5, r \mapsto 1), (r := r * 2; i := i - 1; \texttt{while}\ i > 0\ \texttt{do}\ \ldots) \\
\rightarrow & (n \mapsto 5, i \mapsto 5, r \mapsto 2), (i := i - 1; \texttt{while}\ i > 0\ \texttt{do}\ \ldots) \\
\rightarrow & (n \mapsto 5, i \mapsto 4, r \mapsto 2), (\texttt{while}\ i > 0\ \texttt{do}\ \ldots) \\
\rightarrow\rightarrow\rightarrow & (n \mapsto 5, i \mapsto 3, r \mapsto 4), (\texttt{while}\ i > 0\ \texttt{do}\ (r := r * 2; i := i - 1)) \\
\rightarrow\rightarrow\rightarrow & (n \mapsto 5, i \mapsto 2, r \mapsto 8), (\texttt{while}\ i > 0\ \texttt{do}\ (r := r * 2; i := i - 1)) \\
\rightarrow\rightarrow\rightarrow & (n \mapsto 5, i \mapsto 1, r \mapsto 16), (\texttt{while}\ i > 0\ \texttt{do}\ (r := r * 2; i := i - 1))
\end{array}
$$

Sémantique opérationnelle : exemple

$$
\begin{array}{ll}
 & (n \mapsto 5), (i := n; r := 1; \texttt{while}\ i > 0\ \texttt{do}\ (r := r * 2; i := i - 1)) \\
\rightarrow & (n \mapsto 5, i \mapsto 5), (r := 1; \texttt{while}\ i > 0\ \texttt{do}\ (r := r * 2; i := i - 1)) \\
\rightarrow & (n \mapsto 5, i \mapsto 5, r \mapsto 1), (\texttt{while}\ i > 0\ \texttt{do}\ (r := r * 2; i := i - 1)) \\
\rightarrow & (n \mapsto 5, i \mapsto 5, r \mapsto 1), (r := r * 2; i := i - 1; \texttt{while}\ i > 0\ \texttt{do}\ \ldots) \\
\rightarrow & (n \mapsto 5, i \mapsto 5, r \mapsto 2), (i := i - 1; \texttt{while}\ i > 0\ \texttt{do}\ \ldots) \\
\rightarrow & (n \mapsto 5, i \mapsto 4, r \mapsto 2), (\texttt{while}\ i > 0\ \texttt{do}\ \ldots) \\
\rightarrow\rightarrow\rightarrow & (n \mapsto 5, i \mapsto 3, r \mapsto 4), (\texttt{while}\ i > 0\ \texttt{do}\ (r := r * 2; i := i - 1)) \\
\rightarrow\rightarrow\rightarrow & (n \mapsto 5, i \mapsto 2, r \mapsto 8), (\texttt{while}\ i > 0\ \texttt{do}\ (r := r * 2; i := i - 1)) \\
\rightarrow\rightarrow\rightarrow & (n \mapsto 5, i \mapsto 1, r \mapsto 16), (\texttt{while}\ i > 0\ \texttt{do}\ (r := r * 2; i := i - 1)) \\
\rightarrow\rightarrow\rightarrow & (n \mapsto 5, i \mapsto 0, r \mapsto 32), (\texttt{while}\ i > 0\ \texttt{do}\ (r := r * 2; i := i - 1))
\end{array}
$$

Sémantique opérationnelle : exemple

$$
\begin{array}{cl}
 & (n \mapsto 5), (i := n; r := 1; \texttt{while}\ i > 0\ \texttt{do}\ (r := r * 2; i := i - 1)) \\
\rightarrow & (n \mapsto 5, i \mapsto 5), (r := 1; \texttt{while}\ i > 0\ \texttt{do}\ (r := r * 2; i := i - 1)) \\
\rightarrow & (n \mapsto 5, i \mapsto 5, r \mapsto 1), (\texttt{while}\ i > 0\ \texttt{do}\ (r := r * 2; i := i - 1)) \\
\rightarrow & (n \mapsto 5, i \mapsto 5, r \mapsto 1), (r := r * 2; i := i - 1; \texttt{while}\ i > 0\ \texttt{do}\ \ldots) \\
\rightarrow & (n \mapsto 5, i \mapsto 5, r \mapsto 2), (i := i - 1; \texttt{while}\ i > 0\ \texttt{do}\ \ldots) \\
\rightarrow & (n \mapsto 5, i \mapsto 4, r \mapsto 2), (\texttt{while}\ i > 0\ \texttt{do}\ \ldots) \\
\rightarrow\rightarrow\rightarrow & (n \mapsto 5, i \mapsto 3, r \mapsto 4), (\texttt{while}\ i > 0\ \texttt{do}\ (r := r * 2; i := i - 1)) \\
\rightarrow\rightarrow\rightarrow & (n \mapsto 5, i \mapsto 2, r \mapsto 8), (\texttt{while}\ i > 0\ \texttt{do}\ (r := r * 2; i := i - 1)) \\
\rightarrow\rightarrow\rightarrow & (n \mapsto 5, i \mapsto 1, r \mapsto 16), (\texttt{while}\ i > 0\ \texttt{do}\ (r := r * 2; i := i - 1)) \\
\rightarrow\rightarrow\rightarrow & (n \mapsto 5, i \mapsto 0, r \mapsto 32), (\texttt{while}\ i > 0\ \texttt{do}\ (r := r * 2; i := i - 1)) \\
\rightarrow & (n \mapsto 5, i \mapsto 0, r \mapsto 32), \texttt{skip}
\end{array}
$$

Sémantique opérationnelle : exemple

$$
\begin{array}{cl}
 & (n \mapsto 5), (i := n; r := 1; \texttt{while}\ i > 0\ \texttt{do}\ (r := r * 2; i := i - 1)) \\
\rightarrow & (n \mapsto 5, i \mapsto 5), (r := 1; \texttt{while}\ i > 0\ \texttt{do}\ (r := r * 2; i := i - 1)) \\
\rightarrow & (n \mapsto 5, i \mapsto 5, r \mapsto 1), (\texttt{while}\ i > 0\ \texttt{do}\ (r := r * 2; i := i - 1)) \\
\rightarrow & (n \mapsto 5, i \mapsto 5, r \mapsto 1), (r := r * 2; i := i - 1; \texttt{while}\ i > 0\ \texttt{do}\ \ldots) \\
\rightarrow & (n \mapsto 5, i \mapsto 5, r \mapsto 2), (i := i - 1; \texttt{while}\ i > 0\ \texttt{do}\ \ldots) \\
\rightarrow & (n \mapsto 5, i \mapsto 4, r \mapsto 2), (\texttt{while}\ i > 0\ \texttt{do}\ \ldots) \\
\rightarrow\rightarrow\rightarrow & (n \mapsto 5, i \mapsto 3, r \mapsto 4), (\texttt{while}\ i > 0\ \texttt{do}\ (r := r * 2; i := i - 1)) \\
\rightarrow\rightarrow\rightarrow & (n \mapsto 5, i \mapsto 2, r \mapsto 8), (\texttt{while}\ i > 0\ \texttt{do}\ (r := r * 2; i := i - 1)) \\
\rightarrow\rightarrow\rightarrow & (n \mapsto 5, i \mapsto 1, r \mapsto 16), (\texttt{while}\ i > 0\ \texttt{do}\ (r := r * 2; i := i - 1)) \\
\rightarrow\rightarrow\rightarrow & (n \mapsto 5, i \mapsto 0, r \mapsto 32), (\texttt{while}\ i > 0\ \texttt{do}\ (r := r * 2; i := i - 1)) \\
\rightarrow & (n \mapsto 5, i \mapsto 0, r \mapsto 32), \texttt{skip}
\end{array}
$$

En effet on avait prouvé :

$$\{n \geqslant 0\}\ i := n; r := 1; \texttt{while}\ i > 0\ \texttt{do}\ (r := r * 2; i := i - 1)\ \{r = 2^n\}$$

Théorème (Correction de la logique de Hoare)

Si $\{P\}\ s\ \{Q\}$ alors pour tout m et m', si $P(m)$ et que

$$m, s \quad \longrightarrow\!\!\!\!\rightarrow \cdots \longrightarrow \quad m', \texttt{skip}$$

alors $Q(m')$.

Théorème (Correction de la logique de Hoare)

Si $\{P\}\ s\ \{Q\}$ alors pour tout m et m', si $P(m)$ et que

$$m, s \quad \longrightarrow\!\!\!\!\rightarrow \cdots \rightarrow \quad m', \texttt{skip}$$

alors $Q(m')$.

Remarque :
la réciproque s'appelle la "complétude" n'est pas vraiment prouvable, mais on a ce qu'on appelle la "complétude relative".

Prouver les règles ? Correction de la logique de Hoare

Théorème (Correction de la logique de Hoare)

Si $\{P\}\ s\ \{Q\}$ alors pour tout m et m', si $P(m)$ et que

$$m, s \quad \longrightarrow\!\!\!\!\rightarrow \cdots \longrightarrow \quad m', \texttt{skip}$$

alors $Q(m')$.

Remarque :
la réciproque s'appelle la "complétude" n'est pas vraiment prouvable, mais on a ce qu'on appelle la "complétude relative".

Plus intéressant en général : même chose pour des langages avec des sémantiques plus compliquées (C, Rust, OCaml...)

A-t-on la bonne sémantique opérationnelle ?

Et la question a-t-elle du sens ?

A-t-on la bonne sémantique opérationnelle ?

Et la question a-t-elle du sens ? Oui !

C'est le **compilateur** ou l'**interpréteur** qui donne sens au programme.

A-t-on la bonne sémantique opérationnelle ?

Et la question a-t-elle du sens ? Oui !

C'est le **compilateur** ou l'**interpréteur** qui donne sens au programme.
Il existe des compilateurs vérifiés pour le langage C (CompCert, Vellvm)

A-t-on la bonne sémantique opérationnelle ?

Et la question a-t-elle du sens ? Oui !

C'est le **compilateur** ou l'**interpréteur** qui donne sens au programme.
Il existe des compilateurs vérifiés pour le langage C (CompCert, Vellvm)

Théorème : si $s \rightarrow_C^* \texttt{skip}$ dans la sémantique de C alors
$\text{compile}(s) \rightarrow_{ASM}^* \texttt{skip}$ dans la sémantique de l'assembleur.

Preuves à base de **co**induction.

A-t-on la bonne sémantique opérationnelle ?

Et la question a-t-elle du sens ? Oui !

C'est le **compilateur** ou l'**interpréteur** qui donne sens au programme.
Il existe des compilateurs vérifiés pour le langage C (CompCert, Vellvm)

Théorème : si $s \rightarrow_C^* \texttt{skip}$ dans la sémantique de C alors
$\text{compile}(s) \rightarrow_{ASM}^* \texttt{skip}$ dans la sémantique de l'assembleur.

Preuves à base de **co**induction.

...

A-t-on la bonne sémantique opérationnelle... pour l'assembleur ?
- spécifications très précises donc erreurs rares (et chères)
- quelques formalisations
- tests (cf. modèles mémoire faibles $\approx$ physique expérimentale)

Extensions

Logique de séparation (John Reynolds, 2002)

- motivation : aliasing. En C si x et y sont "aliasé" alors `*y = 1; *x = *y + 1` peut donner $y \mapsto 2$

Extensions

Logique de séparation (John Reynolds, 2002)

- motivation : aliasing. En C si x et y sont "aliasé" alors `*y = 1; *x = *y + 1` peut donner $y \mapsto 2$
- nouveaux connecteurs logiques ! (Bunched Logic)

$P \star Q$ étoile (conjonction séparante)
$P \mathrel{-\!\!\star} Q$ baguette magique (implication séparante)

Extensions

Logique de séparation (John Reynolds, 2002)

- motivation : aliasing. En C si x et y sont "aliasé" alors `*y = 1; *x = *y + 1` peut donner $y \mapsto 2$
- nouveaux connecteurs logiques ! (Bunched Logic)

$$\begin{array}{rl} P \star Q & \text{étoile (conjonction séparante)} \\ P \mathrel{-\!\!\star} Q & \text{baguette magique (implication séparante)} \end{array}$$

Logique de séparation concurrente (Peter O'Hearn, 2007)

$$\frac{\{P_1\}\ s_1\ \{Q_1\} \qquad \{P_2\}\ s_2\ \{Q_2\}}{\{P_1 \star P_2\}\ s_1 || s_2\ \{Q_1 \star Q_2\}}$$

Extensions

Logique de séparation (John Reynolds, 2002)

- motivation : aliasing. En C si x et y sont "aliasé" alors `*y = 1; *x = *y + 1` peut donner $y \mapsto 2$
- nouveaux connecteurs logiques ! (Bunched Logic)

$$\begin{array}{rl} P \star Q & \text{étoile (conjonction séparante)} \\ P \mathrel{-\!\!\star} Q & \text{baguette magique (implication séparante)} \end{array}$$

Logique de séparation concurrente (Peter O'Hearn, 2007)

$$\frac{\{P_1\}\ s_1\ \{Q_1\} \qquad \{P_2\}\ s_2\ \{Q_2\}}{\{P_1 \star P_2\}\ s_1 || s_2\ \{Q_1 \star Q_2\}}$$

Logique de séparation $\approx$ **logique de ressources** complexité en temps, en espace, fantômes, fichiers...

Extensions

Logique de séparation (John Reynolds, 2002)

- motivation : aliasing. En C si x et y sont "aliasé" alors `*y = 1; *x = *y + 1` peut donner $y \mapsto 2$
- nouveaux connecteurs logiques ! (Bunched Logic)

$$\begin{array}{rl} P \star Q & \text{étoile (conjonction séparante)} \\ P \mathrel{-\!\!\star} Q & \text{baguette magique (implication séparante)} \end{array}$$

Logique de séparation concurrente (Peter O'Hearn, 2007)

$$\frac{\{P_1\}\ s_1\ \{Q_1\} \qquad \{P_2\}\ s_2\ \{Q_2\}}{\{P_1 \star P_2\}\ s_1 || s_2\ \{Q_1 \star Q_2\}}$$

Logique de séparation $\approx$ **logique de ressources** complexité en temps, en espace, fantômes, fichiers...
Formalisations en Coq pour C, pour C concurrent, pour Rust, pour OCaml, pour les types, beaucoup de nouvelles choses

Pointeurs

Pour apprendre Coq : **Software Foundations**

- site : https://softwarefoundations.cis.upenn.edu/
- sans installer Coq : https://jscoq.github.io/ext/sf/

Pointeurs

Pour apprendre Coq : **Software Foundations**

- site : https://softwarefoundations.cis.upenn.edu/
- sans installer Coq : https://jscoq.github.io/ext/sf/

Plus jouer avec les invariants : **Why3**

- dans votre navigateur : https://why3.lri.fr/try/

Pointeurs

Pour apprendre Coq : **Software Foundations**

- site : https://softwarefoundations.cis.upenn.edu/
- sans installer Coq : https://jscoq.github.io/ext/sf/

Plus jouer avec les invariants : **Why3**

- dans votre navigateur : https://why3.lri.fr/try/

Attention, c'est addictif !

Pointeurs

Pour apprendre Coq : **Software Foundations**

- site : https://softwarefoundations.cis.upenn.edu/
- sans installer Coq : https://jscoq.github.io/ext/sf/

Plus jouer avec les invariants : **Why3**

- dans votre navigateur : https://why3.lri.fr/try/

Attention, c'est addictif !

Autres exemples dans d'autres systèmes :

- micronoyau L4.verified, en Isabelle
- métro 14, avec Atelier B
- contrôle de vol de l'A380, avec Astrée
- infer (logique de séparation) chez Facebook

Trouvez l'erreur ! (bénigne)